# IRAF講習会
# ～CL Script入門～

国立天文台
天文データセンター
古荘　玲子

2011年2月16日

# 謝辞

過去のADC講習会テキスト（特に、大藪さん、吉田さん）のテキストと、2010年度第3回IRAFソフト開発講習会（2日目）の板さんのテキストを、非常に参考にさせていただきました。有難うございました。



# 凡例

- IRAF(ecl)コマンドラインでの入力

```
ecl> !ds9 &
```

- ファイルの内容（編集すべき内容）

```
print("imcopy")
```

- その他

（IRAFタスクのcompute mode記載例。動作確認のため、eclでコマンドとして入力してもよいだろう）

```
display("hoge.fits", 1, ztrans="log")
```

# なぜIRAFを使うか

メリット

- 無料（重要）
- 多くの人が使っている（重要）
- 豊富な解析タスク・パッケージ
  - タスクを知ってさえいれば大抵のことはできる

デメリット

- ドキュメントがない（に等しい）
- メモリを上手に使えない（らしい）
  私はあまり困ったことがない。←大作を作ったことがないから(^^;;;
- スクリプト開発では・・・
  - デバッガがない。エラーは大抵、意味不明



# CL-script？

eclで必要なタスクを利用すれば、充分解析出来るでしょ。

でも・・・

- いつもいつも同じ手順でやる作業ならば、スクリプトを組んだ方が効率がいい
- 自分の開発した（カワイイ）装置には、解析パイプラインも用意したい
- IRAFの豊富な解析タスクを利用して少しは楽をしたい（ほんと？）

# CL-Script

- IRAFのcl（command language）で動かせるスクリプト言語
- １つのファイルに１つのスクリプト（関数）を記述する。
  - C言語のように１つのファイルにいくつも関数を記述できる訳ではない。
- デバッガは無いので覚悟しましょう
  - えっと思うような些細な事でエラーを吐く
  - エラーの場所がわかりにくい



# Reference

- An Introductory User's Guide to IRAF Scrips
  - http://iraf.noao.edu/iraf/ftp/iraf/docs/script.pdf
  - 古い（V2.8用）が今でも有効
- IRAF CL Script Tips & Tricks
  - http://iraf.noao.edu/iraf/ftp/iraf/docs/script_intro.pdf
  - 比較的新しい（2003年）。プレゼンファイル。例が豊富
- Host CL Scripting Capability
  - http://iraf.noao.edu/iraf/web/new_stuff/cl_host.html
  - CLを外から利用する方法
- help language
- IRAF開発講習会シリーズの過去のテキスト(ADC, NAOJ)
  - http://www.adc.nao.ac.jp/J/cc/public/koshu_shiryo.html#iraf_prog

# 本日の講習の流れ

0. 準備
1. 簡易CLスクリプト
2. CLスクリプトの基本
3. CLスクリプトの機能
4. 実習：解析スクリプトを作ってみよう

余裕があれば・・・

5. さらに先へ



# 0. 準備をしよう

A) ワークディレクトリ（作業環境）と、必要な物を用意しよう

配布物

1. img/KCC*.fits
   実習でつかうFITSデータ
2. img2/KCD*.fits
   実習でつかうFITSデータ（自習用）
3. loginuser.cl
   ユーザ用の環境設定ファイル
4. printit.cl, prompttest.cl, taroex*.cl
   実習用サンプルスクリプト
5. ans/*.cl
   実習課題の解答例
6. panda.cl
   パッケージ化プログラム（自習用）

# 0. 準備をしよう

## B) IRAFの環境を準備しよう

1. mkiraf を実行
2. loginuser.cl を作成しよう（※）

(※) login.clで読み込まれるユーザ環境を記述した設定ファイル

自分の環境設定にlogin.cl自体を変更するかloginuser.clを利用するかは、趣味の問題（だと思う）。

なお、今回は配布されたサンプルスクリプトの中にあります。とりあえず、中身を眺めてみてください。環境準備の時点で重要（かもしれない）設定は以下の3行です

```
reset imextn = "oif:imh fxf:fits,fit,fts,FIT,FTS"
reset imtype = fits
reset stdimage=imt4096
```



# 0. Magic Word

- 動作がおかしい時は・・・
  - unlearn

    CLでパラメータキャッシュをクリアする。task登録したCLスクリプトを書いている途中にパラメータを変えたら、必ずunlearnせよ。
  - flprcache

    プロセスの使ったキャッシュを掃除する。
- それでもダメなら・・・あきらめる

  ➔一旦logoutしてcl再起動

# 1. 簡易CLスクリプト

- タスクを羅列してもCLスクリプトとして利用できる
  （cf. コマンドを羅列したシェルスクリプト）
- パラメータなし型のCLスクリプト

```
ecl> vi taroex0.cl ←taroex0.cl を作成する

      print("imcopy")                          内容
      imcopy img/KCC00003.fits test.fits ↲
                                  改行し忘れに注意

ecl> task $taro0 = taro0.cl ←タスクとして登録
ecl> taro0 ←― $ を頭につける（パラメータなし型の場合）
```



# タスクの記述

- command modeとcompute mode
- command mode

```
ecl> taskname arg1 ... argN par1=val1 par2=val2 ...
```

- compute mode

```
taskname(arg1, ..., argN, par1=val1, par2=val2 ...)
```

# スクリプトのお作法など

- 1タスクは1行に書く

  1つのタスクについて、パラメータが多くて長い記述を2行にわたって書きたい場合は、＼(バックスラッシュ)で改行コードをエスケープする

- タスクはcompute modeで記述した方が良いようです

  なぜか・・・

- 最後の行にも改行を入れる【重要】

  最後の行に改行がないために動かない！ということが、意外とあります



# mkscript

- 簡易CLスクリプトを作成するタスク

```
ecl> mkscript
Script file name (scr.cl):  tarotmp.cl
Task name of command to be added to script: imcopy
```

このあと、imcopyのepar画面が出る。
適当に編集して、:wqで抜ける。

```
Is the command ok? (yes): yes
Add another command? (yes): no
Is the script ok? (yes): yes
Submit the script as a background job? (yes): yes
```

←他にタスクを足したかったらyesと入力して繰り返し

CLスクリプトの作成例が欲しい時に便利・・・

# 2. CLスクリプトの基本

基本構成（パラメータあり型）

1. procedure宣言
2. 明示パラメータ（スクリプト引数）宣言
3. 隠しパラメータ宣言
4. list directed parameters宣言
5. begin
6. スクリプトの中身
7. end



# パラメータ(変数)のデータ型

int　　整数型 32bit
real　　実数型。指数はEで表す ex.) 2.5E+2
bool　　bool型(真偽値) yes / no
　　真＝1, 偽＝0
string　　文字列型
file　　ファイル型(stringと変わらないらしい？)
struct　　特殊な文字列型
　　空白も文字列の一部として扱う
gcur, imcur　カーソルパラメータ

# サンプルコード：printit.cl

```
procedure printit (file_name)
string file_name
struct *flist
begin
    struct line
    flist = file_name
    while( fscan( flist, line ) != EOF )
        print(line)
end
```

← 始まりはprocedure

← スクリプト引数の宣言

これは隠しパラメータ

← list directed parameterの宣言

スクリプトの始まり

これは内部変数

スクリプトの終わり



# printit.clを実行してみる

- タスクの登録

```
ecl> task printit = printit.cl
```

パラメータあり型の場合、頭に$はつけない

- 実行

```
ecl> printit printit.cl
```

printit.cl自身を表示させてみる

# パラメータの読み取り

```
procedure printit (file_name)      ← 始まりはprocedure
string file_name                   ← スクリプト引数の宣言
struct *flist   これは隠しパラメータ   ← list directed parameterの宣言
begin   スクリプトの始まり
    struct line   これは内部変数
    flist = file_name
    while( fscan( flist, line ) != EOF )
        print(line)
end   スクリプトの終わり
```



# パラメータの読み取り

- **scan ( p1, p2, … )**
  - **標準入力から読み取って内部変数に格納**

```
ecl> string ss1
ecl> =scan( ss1 )
        何か文字列を打ち込む。
ecl> =ss1
```

- **fscan( pp, p1, p2, … )**
  - **内部変数ppから読み取って別の内部変数に格納**

```
ecl> string ss2
ecl> =fscan( ss1, ss2 )
ecl> =ss2
```

# stringとstructの違い

```
ecl> string origstr = “Hello world”
ecl> string ss3
ecl> struct st1
ecl> =fscan ( origstr, ss3 )
ecl> =ss3
ecl> =fscan ( origstr, st1 )
ecl> =st1
```



# 3. CLスクリプトの機能

1. prompting
2. mktemp
3. list directed parameters (LDP)
4. ワイルドカードの取り扱いとsections

# prompting

- 実行時にパラメータへの入力を促す
- パラメータ宣言のところに記述

propmttest.cl

```
procedure prompttest (file_name)
string file_name {prompt = "Input file name"}
struct *flist
begin
    struct line
    flist = file_name
     :
end
```

プロンプトのメッセージ内容

← 内部変数へ代入されるときにプロンプトが出る



# mktemp

- 一時使用ファイル（temporary file）を作るコマンド
- ファイル名を自動生成する
- スクリプト中で大変便利
- 片づけるのを忘れずに
- 例）

```
ecl> tmpfile = mktemp( "hoge." )
```

➔ hoge.xxxx（xxxxは数字）というファイル名が自動生成されてtmpfileにアサインされる。

# list directed parameters(LDP)

- テキストファイルの中身を、改行で区切られた文字列の順序リストとして格納してくれる
- CLスクリプト中での宣言：必ずbeginの前で、

  `struct *ppp`　あるいは　`string *ppp`
- テキストファイルをLDPに格納するやり方

  `ppp = file_name`
- CLスクリプト中でファイルの中身を順序読み出しするときは、必ずこれを使う。



# ワイルドカードとsectionsコマンド

- ワイルドカード
  - UNIXワイルドカード（*, ?, …）
  - IRAFの@付ファイルリスト
- sectionsコマンド
  - ワイルドカードはsectionsコマンドで展開して標準出力に書き出す
  - 例）

```
sections( “*.fits”, option=“full” )
```

```
sections( “@list”, option=“root” )
```

# LDPとsectionsを組み合わせてワイルドカードを展開してファイルを読む

```
struct *flist
string infile, junk
tmpfile = mktemp("hoge.")
sections( infile, option="full", > tmpfile)
flist = tmpfile
while( fscan( flist, junk ) != EOF )
  { ・・・ }
```

infileに格納されたワイルドカードを展開して、tmpfileにファイル名リストとして入れる

tmpfileをLDPに格納

fscanでLDPから一つずつファイル名をjunkに読み込んでいく



# 4. 実習

実習のサンプルスクリプトプログラムのタスク登録と実行例）

```
ecl> task taroex1 = taroex1.cl
ecl> taroex1
```

とりあえず、サンプルスクリプトを眺め、実行してみよう

# FITS画像のヘッダをよむ

FITS画像リスト(img/KCC*.fits)を入力して、FITSファイル名と天体名(OBJECT)のリストを出力する（taroex1.cl）

➔FITSヘッダを読むには？

–使うタスク：imgets

```
while( fcan( flist, inname ) != EOF ) {
    imgets( inname, 'title' )
    obj = imgets.value
}
```



# 課題（1）

taroex1.cl を改訂して、FITS画像リスト(img/KCC*.fits)を入力して、FITSファイル名、天体名、露出時間、画像の平均値(mean)、画像のモード(mode)を出力するスクリプトを作成しなさい

➔画像の平均値等？ ファイル名iminfo.clで作成

–使うタスク：imgets, imstat

出力例

```
img/KCC00003.fits (1943)Anteros 300.0 7311.04 7250.83
img/KCC00004.fits (1943)Anteros 300.0 7289.13 7227.56
:
```

# タスクの結果をscanに渡す(パイプ)

```
real mn, md
        :
imstat( img, field="mean, mode", format-) | \
 scan (mn, md)
```

1タスクを2行にわたって書きたいときは
\(バックスラッシュ)で行末をエスケープする

Imstat で img の mean, mode を表示
⇒パイプで表示結果を scan に渡す
scan はパイプで渡された値を mn, md に格納



# FITSデータを仕分ける

FITS画像のヘッダキーワードDATA-TYPをもとに画像ファイルを分類し、リストファイル（テキスト）に書き出す（taroex2.cl）

➔FITSヘッダの情報を得るには？

- –使うタスク：hselect (例えば)
  ※imgetsでも可能
- –結果をファイルに出力する

```
outfiles = datatyp//"."//title //は文字列連結
print( param, >> outfiles )
```

# 標準入力を読み取る

標準入力の文字列を読み取ってbool型変数に入れる（taroex3.cl）

➔ nが入力されるまで繰り返す

y か n 以外は無視される

```
bool flg = yes
While( flg ){
  print( “--> Continue? <y/n>” )
  while( scan( flg ) == 0 ) {
  }
  if ( ! flg ) break
  else print( flg//“ Continue.” )
}
bye
```



# 画像をds9に表示させる

画像を指定してds9に表示（taroex4.cl）

–使うタスク：display

```
ecl> !ds9 &
```

```
String image_name {prompt = “Input image name”}

begin
    string filename
    filename = image_name
    display(filename, 1, contras=0.0005,
ztrans = “log”, nsample=100000)
end
```

実際は1行で書く

# 隠しパラメータを変更可能に

## 隠しパラメータを実行時に変更できるようにする：taroex4n.cl

displayタスクのパラメータztrans="linear"も選べるように

```
String image_name {prompt = "Input image name"}
String ztrans="log" {prompt = "ztrans for display",
enum="log|linear"}

begin
    string filename,zt
    filename = image_name
    zt = ztrans
    display(filename, 1, contras=0.0005, ztrans = zt,
nsample=100000)
end
```

実際は1行で書く

実際は1行で書く



# 課題（2）

① taroex1.cl と taroex4.cl を組み合わせて、FITS画像リスト(img/KCC0*.fits)を入力して画像を順にds9に表示するスクリプトを作成しなさい

② さらに、(taroex3.cl を参考にして)標準入力からの入力で、次の画像を表示するタイミングを制御できるようにしなさい（yを入力すると次の画像を表示する）

seqdisplay.cl

# ds9の座標値を読み取る

ds9のカーソルの座標値をfscanで読み取って変数に入れる（taroex5.cl）

➔カーソル変数imcurを使う

➔ds9は起動しておくこと

```
While( fscan( imcur, xx, yy, wcs, command ) != EOF ){
    key = substr( command, 1, 1 )
    if ( key == “q” ){  ← カーソル上での入力が q ならば抜ける
        break
    } else {
        print( xx, yy )
    }
}
```



# 課題（3）

ds9上で画像の左下と右上2点を指定して、その矩形領域の統計情報（ピクセル数、mean、modeなど）を表示するスクリプトを作成しなさい

- 使うタスク： imstatistics

ファイル名imreqtstat.clで作成

【余裕がある人は】スクリプトを、画像1枚ではなく画像リストを引数として入力するように改造しなさい。また、表示した矩形領域および統計情報をファイルに出力するようにしなさい

# インタラクティブな画像位置合わせ

カーソル座標値をfscanで読み取って変数に入れ、reference画像とのshift量を一時ファイルに格納。結果ファイル名リストは事前に作成（taroex6.cl）

```
string tmpfile1, tmpfile2, tmpfile3
tmpfile1 = mktemp( “tmp$shiftimg1.” )
 (...snip...)
while( fscan( flist, fname ) != EOF ) {
    display( fname, 1 )
    if( fscan( imcur, xx, yy ) == 2 ){
        sx = refx – xx
        sy = refy – yy
        print( sx, sy, >> tmpfile3 )
    }
}
imalign( inimg, refimg, tmpfile2, “@”//outlist, shifts=tmpfile3)
```

実行したら、位置合わせ結果を seqdisplay.cl で表示してみましょう。

⇒小惑星は見つかりましたか・・・？

実行のしかたは次ページ



# インタラクティブな画像位置合わせ

準備（imalignの結果ファイルのファイル名リストを作成）

```
ecl> sections(“img/KCC0*%.fits%_sh.fits%”, > “shiftfile.lst”)
```

結果画像のファイル名リストを作る。
%で囲まれた文字列を置換

入力

```
ecl> task taroex6 = taroex6.cl
ecl> taroex6 img/KCC0*.fits img/KCC00003.fits shiftfile.lst
```

reference画像

Imalign結果の画像ファイル名リスト

動作

- 最初に、reference画像として指定した画像をds9に表示
  - ➔ 適当な星(※)にカーソルを合わせて何かキー（“q”以外）を打つと、その位置をreference座標として記録
- 次に、画像リストとして入力した画像(img/KCC0*.fits)を順にds9に表示
  - ➔ カーソルを※と同じ星に合わせてキーを打つ
  - ➔ 今表示している画像の星の座標とreference座標との差分を記録
  - ➔ imalignを実行。結果は引数で与えたファイルリストに記載されているFITSファイルへ

# 課題（4）

taroex6.clをもとに、特定の天体に位置を合わせて画像を合成するスクリプトを作りなさい（ヒント：シフト結果のファイル名もmktempで作成し、終わったら消去する）

- 使うタスク： imalign, imcombine

ファイル名`imshiftcomb.cl`で作成

【余裕がある人は】imalign, imcombineのパラメータを隠しパラメータとして設定したうえで、実行時に変更できるように改造してみよう。



# 5. さらに先へ

# パッケージ登録

- panda.clを参考にして、パッケージを作成してみよう

おまけ

- 自作CLスクリプトをIRAF CLログイン時に常にタスクとして登録するには？

loginuser.cl

```
set mycls = "/full/path/to/clscriptfilesdir/"
task taroex0 = mycls$taroex0.cl
```

CLスクリプトファイルが置かれているディレクトリへの絶対パス



# 解析パイプラインを作ってみよう

- img2/ のデータを解析するパイプラインを作りなさい
  1. FITSヘッダキーワードDATA-TYPで、オブジェクト、フラット、バイアスに分類
  2. バイアスを合成する
  3. オブジェクト・フラットフレームからバイアスを引く
  4. フラットを合成して、オブジェクトフレームをフラット補正する
  5. オブジェクトフレームを位置合わせし合成する（インタラクティブに）※

# その他、知っていると便利なコマンド

- bye
  スクリプトを途中終了する
- next
  繰り返しの制御構造（for, while等）で次ターンとしてループ先頭へ
  - C言語の continue; と同等
- goto
  指定したラベルの行へジャンプ（ラベルは末尾にコロン":"をつける）
- access
  ファイルが存在するかどうか判定（存在すれば真=1）
  - 例

```
if( access( infile ) == 0 ) bye
```



# IRAF外部からCLを利用

- 明日の板さんの講義をお楽しみに！